KNN KD Tree

In [3]:

```
# Load the Drive helper and mount
from google.colab import drive

# This will prompt for authorization.
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6
qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%
b&scope=email%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.
2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.photos.readonly%20https%3A%2F%2Fww
ogleapis.com%2Fauth%2Fpeopleapi.readonly&response_type=code

Enter your authorization code:
··········
Mounted at /content/drive
```

In [0]:

```
%env JOBLIB_TEMP_FOLDER=/tmp
```

```
env: JOBLIB_TEMP_FOLDER=/tmp
```

In [0]:

```
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

# using the SQLite Table to read data.
con = sqlite3.connect('database.sqlite')

#filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 """, con)

# Give reviews with Score>3 a positive rating, and reviews with a score<3 a negative rating.
def partition(x):
    if x < 3:
        return 0
```

```
    return 1

#changing reviews with score less than 3 to be positive and vice-versa
actualScore = filtered_data['Score']
positiveNegative = actualScore.map(partition)
filtered_data['Score'] = positiveNegative
print(filtered_data.shape)
```

In [0]:

```
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind='qui
cksort', na_position='last')
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inpl
ace=False)
final.shape
```

In [0]:

```
final.head(2)
```

In [0]:

```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
#Before starting the next phase of preprocessing lets see the number of entries left
print(final.shape)

#How many positive and negative reviews are present in our dataset?
final['Score'].value_counts()
```

In [0]:

```
final.sort_values('Time',axis=0,ascending=True,inplace=True,kind='quicksort')
final.head(2)
```

In [0]:

```
stop = set(stopwords.words('english')) #set of stopwords
sno = nltk.stem.SnowballStemmer('english') #initialising the snowball stemmer

def cleanhtml(sentence): #function to clean the word of any html-tags
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, ' ', sentence)
    return cleantext
def cleanpunc(sentence): #function to clean the word of any punctuation or special characters
    cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)
    cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
    return  cleaned
```

In [0]:

```
#Code for implementing step-by-step the checks mentioned in the pre-processing phase
# this code takes a while to run as it needs to run on 500k sentences.
i=0
str1=' '
final_string=[]
all_positive_words=[] # store words from +ve reviews here
all_negative_words=[] # store words from -ve reviews here.
s=''
for sent in final['Text'].values:
    filtered_sentence=[]
    #print(sent);
    sent=cleanhtml(sent) # remove HTMl tags
    for w in sent.split():
        for cleaned_words in cleanpunc(w).split():
            if((cleaned_words.isalpha()) & (len(cleaned_words)>2)):
                if(cleaned_words.lower() not in stop):
                    s=(sno.stem(cleaned_words.lower())).encode('utf8')
                    filtered_sentence.append(s)
                    if (final['Score'].values)[i] == 1:
                        all_positive_words.append(s) #list of all words used to describe positive r
eviews
```

```
                    if(final['Score'].values)[i] == 0:
                        all_negative_words.append(s) #list of all words used to describe negative r
eviews reviews
                else:
                    continue
            else:
                continue
    #print(filtered_sentence)
    str1 = b" ".join(filtered_sentence) #final string of cleaned words
    #print("***********************************************************************")

    final_string.append(str1)
    i+=1
```

In [0]:

```
final['CleanedText']=final_string #adding a column of CleanedText which displays the data after pr
e-processing of the review
final['CleanedText']=final['CleanedText'].str.decode("utf-8")
final.head(3)
```

In [0]:

```
import pickle
pickle.dump(final, open('final.p', 'wb'))
#final_sent = pickle.load(open('data.p','rb'))
final.shape
```

In [9]:

```
import pickle
final = pickle.load(open('drive/My Drive/Colab Notebooks/gbdt/final.p','rb'))


from sklearn.model_selection import train_test_split
##Sorting data according to Time in ascending order for Time Based Splitting
time_sorted_data = final.sort_values('Time', axis=0, ascending=True, inplace=False, kind='quicksort
', na_position='last')
final.head(2)
```

Out[9]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summ
138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	0	1	939340800	EVI bo educati
138683	150501	0006641040	AJ46FKXOVC7NR	Nicholas A Mesiano	2	2	1	940809600	This w seri great to sp time

In [28]:

```
y = final['Score'].iloc[:60000]
x = final['CleanedText'].iloc[:60000]
x.shape,y.shape
```

Out[28]:

```
((60000,), (60000,))
```

BOW

In [29]:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer

X_tra, X_tes, y_train, y_test = train_test_split(x,y,test_size=0.3,shuffle=False,random_state=0)

#Implementing BAG of words
bow = CountVectorizer(max_features =2000)
X_tf =bow.fit_transform(X_tra)

# Standerdising the data
norm = StandardScaler(with_mean = False)
X_train = norm.fit_transform(X_tf)

# tfidf test
X_tfte = bow.transform(X_tes)

# Standerdising the data
X_test = norm.transform(X_tfte)


X_train.shape,y_train.shape,X_test.shape,y_test.shape
```

Out[29]:

```
((42000, 2000), (42000,), (18000, 2000), (18000,))
```

In [0]:

```
%env JOBLIB_TEMP_FOLDER=/tmp
```

```
env: JOBLIB_TEMP_FOLDER=/tmp
```

In [0]:

```
# Choosing 500 as max informaton dimenson
from sklearn.decomposition import TruncatedSVD


tsvd= TruncatedSVD(n_components=500)
data=tsvd.fit_transform(X_train)

tsvd1= TruncatedSVD(n_components=500)
data1=tsvd1.fit_transform(X_test)
```

In [31]:

```
tsvd1.components_.shape,tsvd.components_.shape
```

Out[31]:

```
((500, 2000), (500, 2000))
```

In [0]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.cross_validation import cross_val_score
from collections import Counter
from sklearn.metrics import accuracy_score
from sklearn import cross_validation
from sklearn.metrics import confusion_matrix



# 10 fold Cross Validation
myList = list(range(0,14))
neighbors = list(filter(lambda x: x % 2 != 0, myList))
```

In [0]:

```
cv_scores = []

for k in neighbors:
    knn = KNeighborsClassifier(n_neighbors=k,algorithm='kd_tree')
    scores = cross_val_score(knn, data, y_train, cv=10, scoring='f1_weighted')
    print('\nThe cross validation score for K = {} is {}.'.format(k,scores.mean()))
    cv_scores.append(scores.mean())

MSE = [1 - x for x in cv_scores]

optimal_k = neighbors[MSE.index(min(MSE))]
```

```
The cross validation score for K = 1 is 0.8527174820142207.

The cross validation score for K = 3 is 0.8706126607535524.

The cross validation score for K = 5 is 0.869991770063425.

The cross validation score for K = 7 is 0.86598161537646.

The cross validation score for K = 9 is 0.8625663413502307.

The cross validation score for K = 11 is 0.8610484428255735.

The cross validation score for K = 13 is 0.8587271383963613.
```

In [0]:

```
from sklearn.neighbors import KNeighborsClassifier

knn_optimal = KNeighborsClassifier(n_neighbors=3,algorithm='kd_tree')
knn_optimal.fit(data,y_train)

pred = knn_optimal.predict(data1)
```

In [36]:

```
from sklearn.metrics import f1_score
f1 = f1_score(y_test, pred, average='micro')

f1_sc =f1*100

print('\nThe f1 score of the knn classifier of BOW for k = %d is %f%%' % (3, f1_sc))
```

```
The f1 score of the knn classifier of BOW for k = 3 is 86.533333%
```

TFIDF

In [37]:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer

X_tra, X_tes, y_train, y_test = train_test_split(x,y,test_size=0.3,shuffle=False)

#Implementing BAG of words
tfidf = TfidfVectorizer(ngram_range=(0,1),max_features=2000,dtype=float)
X_tf =tfidf.fit_transform(X_tra)

# Standerdising the data
norm = StandardScaler(with_mean = False)
X_train = norm.fit_transform(X_tf)

# tfidf test
X_tfte = tfidf.transform(X_tes)
```

```
# Standerdising the data
X_test = norm.transform(X_tfte)


from sklearn.model_selection import TimeSeriesSplit


X_train.shape,X_test.shape,y_train.shape,y_test.shape
```

Out[37]:

```
((42000, 2000), (18000, 2000), (42000,), (18000,))
```

In [0]:

```
# Choosing 500 as max informaton
from sklearn.decomposition import TruncatedSVD


tsvd= TruncatedSVD(n_components=500)
data=tsvd.fit_transform(X_train)

tsvd1= TruncatedSVD(n_components=500)
data1=tsvd1.fit_transform(X_test)
```

In [39]:

```
tsvd1.components_.shape,tsvd.components_.shape
```

Out[39]:

```
((500, 2000), (500, 2000))
```

In [17]:

```
import warnings
warnings.filterwarnings("ignore")
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.cross_validation import cross_val_score
from collections import Counter
from sklearn.metrics import accuracy_score
from sklearn import cross_validation
from sklearn.metrics import confusion_matrix



# 10 fold Cross Validation
myList = list(range(0,14))
neighbors = list(filter(lambda x: x % 2 != 0, myList))

cv_scores = []

for k in neighbors:
    knn = KNeighborsClassifier(n_neighbors=k,algorithm='kd_tree')
    scores = cross_val_score(knn, data, y_train, cv=10, scoring='f1_weighted')
    print('\nThe cross validation score for K = {} is {}.'.format(k,scores.mean()))
    cv_scores.append(scores.mean())

MSE = [1 - x for x in cv_scores]

optimal_k = neighbors[MSE.index(min(MSE))]
```

```
The cross validation score for K = 1 is 0.8532029141917192.

The cross validation score for K = 3 is 0.8595867075788926.

The cross validation score for K = 5 is 0.8572394189785726.

The cross validation score for K = 7 is 0.8546660306090154.

The cross validation score for K = 9 is 0.8531064367196812.
```

```
The cross validation score for K = 11 is 0.8508529079047541.

The cross validation score for K = 13 is 0.8488066902135725.
```

In [18]:

```
print('\nThe optimal number of neighbors is %d.' % optimal_k)
```

```
The optimal number of neighbors is 3.
```

In [0]:

```
from sklearn.neighbors import KNeighborsClassifier

knn_optimal = KNeighborsClassifier(n_neighbors=3,algorithm='kd_tree')
knn_optimal.fit(data,y_train)

pred = knn_optimal.predict(data1)
```

In [41]:

```
from sklearn.metrics import f1_score
f1 = f1_score(y_test, pred, average='micro')

f1_sc =f1*100

print('\nThe f1 score of the knn classifier of BOW for k = %d is %f%%' % (3, f1_sc))
```

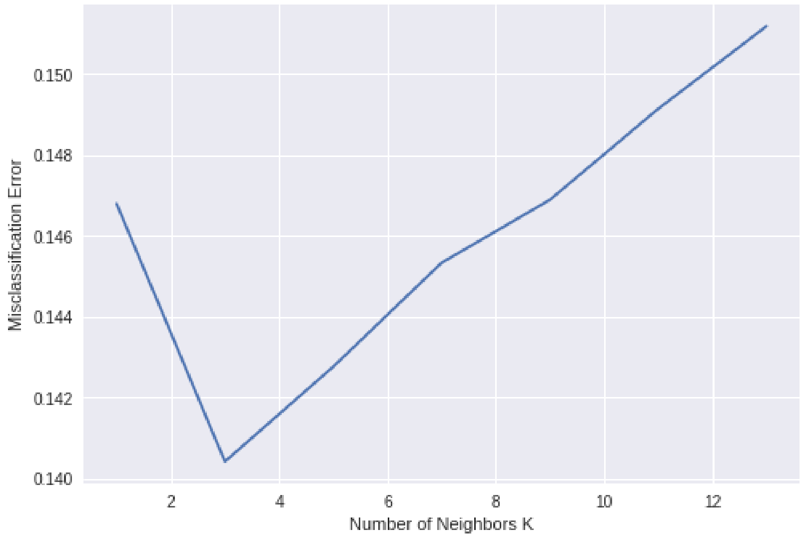
```
The f1 score of the knn classifier of BOW for k = 3 is 87.455556%
```

In [21]:

```
import matplotlib.pyplot as plt
import numpy as np
# plot misclassification error vs k
plt.plot(neighbors, MSE)


plt.xlabel('Number of Neighbors K')
plt.ylabel('Misclassification Error')
plt.show()

print("the misclassification error for each k value is : ", np.round(MSE,3))
```




```
the misclassification error for each k value is :  [0.147 0.14  0.143 0.145 0.147 0.149 0.151]
```

Avg w2c

In [4]:

```
!pip install gensim
```

```
Collecting gensim
  Downloading
https://files.pythonhosted.org/packages/27/a4/d10c0acc8528d838cda5eede0ee9c784caa598dbf40bd0911ff8c
7eb/gensim-3.6.0-cp36-cp36m-manylinux1_x86_64.whl (23.6MB)
    100% |████████████████████████████████| 23.6MB 1.4MB/s
Requirement already satisfied: six>=1.5.0 in /usr/local/lib/python3.6/dist-packages (from gensim)
(1.11.0)
Requirement already satisfied: scipy>=0.18.1 in /usr/local/lib/python3.6/dist-packages (from
gensim) (1.1.0)
Requirement already satisfied: numpy>=1.11.3 in /usr/local/lib/python3.6/dist-packages (from
gensim) (1.14.6)
Collecting smart-open>=1.2.1 (from gensim)
  Downloading
https://files.pythonhosted.org/packages/4b/1f/6f27e3682124de63ac97a0a5876da6186de6c19410feab66c1543
055/smart_open-1.7.1.tar.gz
Collecting boto>=2.32 (from smart-open>=1.2.1->gensim)
  Downloading
https://files.pythonhosted.org/packages/23/10/c0b78c27298029e4454a472a1919bde20cb182dab1662cec7f2ca
523/boto-2.49.0-py2.py3-none-any.whl (1.4MB)
    100% |████████████████████████████████| 1.4MB 12.3MB/s
Collecting bz2file (from smart-open>=1.2.1->gensim)
  Downloading
https://files.pythonhosted.org/packages/61/39/122222b5e85cd41c391b68a99ee296584b2a2d1d233e7ee32b453
f2d/bz2file-0.98.tar.gz
Requirement already satisfied: requests in /usr/local/lib/python3.6/dist-packages (from smart-
open>=1.2.1->gensim) (2.18.4)
Collecting boto3 (from smart-open>=1.2.1->gensim)
  Downloading
https://files.pythonhosted.org/packages/a0/d8/8b000ffeba218d47cd81fbd0bd0b2790742f81ffe116964a298be
8a4/boto3-1.9.50-py2.py3-none-any.whl (128kB)
    100% |████████████████████████████████| 133kB 26.7MB/s
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-packages (from
requests->smart-open>=1.2.1->gensim) (2018.10.15)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /usr/local/lib/python3.6/dist-packages
(from requests->smart-open>=1.2.1->gensim) (3.0.4)
Requirement already satisfied: idna<2.7,>=2.5 in /usr/local/lib/python3.6/dist-packages (from
requests->smart-open>=1.2.1->gensim) (2.6)
Requirement already satisfied: urllib3<1.23,>=1.21.1 in /usr/local/lib/python3.6/dist-packages
(from requests->smart-open>=1.2.1->gensim) (1.22)
Collecting botocore<1.13.0,>=1.12.50 (from boto3->smart-open>=1.2.1->gensim)
  Downloading
https://files.pythonhosted.org/packages/82/c1/a26012b4dbca2e2ae06d7b24dc6b4378c0b0544d25f3e3f216612
033/botocore-1.12.50-py2.py3-none-any.whl (4.9MB)
    100% |████████████████████████████████| 4.9MB 6.6MB/s
Collecting s3transfer<0.2.0,>=0.1.10 (from boto3->smart-open>=1.2.1->gensim)
  Downloading
https://files.pythonhosted.org/packages/d7/14/2a0004d487464d120c9fb85313a75cd3d71a7506955be458eebfe
b1d/s3transfer-0.1.13-py2.py3-none-any.whl (59kB)
    100% |████████████████████████████████| 61kB 22.8MB/s
Collecting jmespath<1.0.0,>=0.7.1 (from boto3->smart-open>=1.2.1->gensim)
  Downloading
https://files.pythonhosted.org/packages/b7/31/05c8d001f7f87f0f07289a5fc0fc3832e9a57f2dbd4d3b0fee70e
365/jmespath-0.9.3-py2.py3-none-any.whl
Collecting docutils>=0.10 (from botocore<1.13.0,>=1.12.50->boto3->smart-open>=1.2.1->gensim)
  Downloading
https://files.pythonhosted.org/packages/36/fa/08e9e6e0e3cbd1d362c3bbee8d01d0aedb2155c4ac112b19ef3ca
d8d/docutils-0.14-py3-none-any.whl (543kB)
    100% |████████████████████████████████| 552kB 23.8MB/s
Requirement already satisfied: python-dateutil<3.0.0,>=2.1; python_version >= "2.7" in
/usr/local/lib/python3.6/dist-packages (from botocore<1.13.0,>=1.12.50->boto3->smart-open>=1.2.1->
gensim) (2.5.3)
Building wheels for collected packages: smart-open, bz2file
  Running setup.py bdist_wheel for smart-open ... -   done
  Stored in directory:
/root/.cache/pip/wheels/23/00/44/e5b939f7a80c04e32297dbd6d96fa3065af89ecf57e2b5f89f
  Running setup.py bdist_wheel for bz2file ... -   done
  Stored in directory:
/root/.cache/pip/wheels/81/75/d6/e1317bf09bf1af5a30befc2a007869fa6e1f516b8f7c591cb9
Successfully built smart-open bz2file
Installing collected packages: boto, bz2file, docutils, jmespath, botocore, s3transfer, boto3, sma
rt-open, gensim
Successfully installed boto-2.49.0 boto3-1.9.50 botocore-1.12.50 bz2file-0.98 docutils-0.14 gensim
```

```
-3.6.0 jmespath-0.9.3 s3transfer-0.1.13 smart-open-1.7.1
```

In [10]:

```python
y = final['Score'].iloc[:50000]
x = final['CleanedText'].iloc[:50000]
x.shape,y.shape
```

Out[10]:

```
((50000,), (50000,))
```

In [0]:

```python
X_tra, X_tes, y_train, y_test = train_test_split(x,y,test_size=0.3,shuffle=False)

sent_of_train=[]
for sent in X_tra:
    sent_of_train.append(sent.split())

sent_of_test=[]
for sent in X_tes:
    sent_of_test.append(sent.split())
```

In [12]:

```python
#word to vector
from gensim.models import Word2Vec
w2v_model=Word2Vec(sent_of_train,min_count=3,size=200, workers=4)# words which occurs 3 times; 500 dimensions

w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 3 times ",len(w2v_words))
```

```
number of words that occured minimum 3 times  10559
```

In [13]:

```python
# compute average word2vec for each review for X_train .
from tqdm import tqdm
import numpy as np



train_vectors = []
for sent in tqdm(sent_of_test):
    sent_vec = np.zeros(200)
    cnt_words =0;
    for word in sent:
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    train_vectors.append(sent_vec)
```

```
100%|██████████| 15000/15000 [00:20<00:00, 732.50it/s]
```

In [14]:

```python
train_vectors1 = []
for sent in tqdm(sent_of_train):
    sent_vec = np.zeros(200)
    cnt_words =0;
    for word in sent:
        if word in w2v_words:
            vec = w2v_model.wv[word]
```

```
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    train_vectors1.append(sent_vec)
```

```
100%|██████████| 35000/35000 [00:45<00:00, 762.87it/s]
```

In [15]:

```
len(train_vectors),len(train_vectors1)
```

Out[15]:

```
(15000, 35000)
```

In [16]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import TimeSeriesSplit

# Data-preprocessing: Standardizing the data
sc = StandardScaler(with_mean = False)
X_train3 = sc.fit_transform(train_vectors1)
X_test3 = sc.transform(train_vectors)

tscv = TimeSeriesSplit(n_splits=4)


y_train.shape,X_train3.shape,X_test3.shape
```

Out[16]:

```
((35000,), (35000, 200), (15000, 200))
```

In [0]:

```
pickle.dump(X_train3, open('X_train3.p', 'wb'))
pickle.dump(X_test3, open('X_test3.p', 'wb'))
```

In [0]:

```
#############################
X_train3 = pickle.load(open('drive/My Drive/Colab Notebooks/gbdt/X_train3.p','rb'))
X_test3 = pickle.load(open('drive/My Drive/Colab Notebooks/gbdt/X_test3.p','rb'))
```

In [52]:

```
########################################################

from sklearn.neighbors import KNeighborsClassifier
from sklearn.cross_validation import cross_val_score
from sklearn.metrics import make_scorer
from sklearn.metrics import f1_score


# 10 fold Cross Validation
myList = list(range(0,15))
neighbors = list(filter(lambda x: x % 2 != 0, myList))

cv_scores = []

for k in neighbors:
    knn = KNeighborsClassifier(n_neighbors=k,algorithm='kd_tree')
    scores = cross_val_score(knn,X_train3,y_train, cv=10, scoring='f1_weighted')
    print('\nThe cross validation score for K = {} is {}.'.format(k,scores.mean()))
    cv_scores.append(scores.mean())

MSE = [1 - x for x in cv_scores]

optimal_k = neighbors[MSE.index(min(MSE))]
```

```
The cross validation score for K = 1 is 0.857318544776039.

The cross validation score for K = 3 is 0.8726487714379493.

The cross validation score for K = 5 is 0.8750511773281786.

The cross validation score for K = 7 is 0.8741614135709576.

The cross validation score for K = 9 is 0.873172685450802.

The cross validation score for K = 11 is 0.8722327409519626.

The cross validation score for K = 13 is 0.8712856412673456.
```

In [0]:

```
from sklearn.neighbors import KNeighborsClassifier
knn_optimal = KNeighborsClassifier(n_neighbors=5,algorithm='kd_tree')
```

In [0]:

```
knn_optimal.fit(X_train3,y_train)

pred = knn_optimal.predict(X_test3)
```

In [20]:

```
from sklearn.metrics import f1_score
f1 = f1_score(y_test, pred, average='micro')

f1_sc =f1*100

print('\nThe f1 score of the knn classifier of BOW for k = %d is %f%%' % (5, f1_sc))
```

```
The f1 score of the knn classifier of BOW for k = 5 is 88.820000%
```

AVG TF-IDF W2V

In [0]:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer

X_tra, X_tes, y_train, y_test = train_test_split(x.values,y.values,test_size=0.3,random_state=0)
```

In [0]:

```
sent_of_train=[]
for sent in X_tra:
    sent_of_train.append(sent.split())

sent_of_test=[]
for sent in X_tes:
    sent_of_test.append(sent.split())
```

In [23]:

```
#word to vector
from gensim.models import Word2Vec
w2v_model=Word2Vec(sent_of_train,min_count=3,size=200, workers=4)# words which occurs 3 times; 500
dimensions

w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 3 times ",len(w2v_words))
```

```
number of words that occured minimum 3 times  10497
```

In [0]:

```python
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
m = TfidfVectorizer()
tf_idf_matrix = m.fit_transform(X_tra)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(m.get_feature_names(), list(m.idf_)))
```

In [25]:

```python
# TF-IDF weighted Word2Vec
tfidf_feat = m.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(sent_of_train): # for each review/sentence
    sent_vec = np.zeros(200) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████| 35000/35000 [00:50<00:00, 694.03it/s]
```

In [26]:

```python
tfidf_sent_vectors1 = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(sent_of_test): # for each review/sentence
    sent_vec = np.zeros(200) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors1.append(sent_vec)
    row += 1
```

```
100%|██████████| 15000/15000 [00:21<00:00, 695.51it/s]
```

In [0]:

```python
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import TimeSeriesSplit

# Data-preprocessing: Standardizing the data
sc = StandardScaler(with_mean = False)
X_train4 = sc.fit_transform(tfidf_sent_vectors)
X_test4 = sc.transform(tfidf_sent_vectors1)
```

In [28]:

```python
X_train4.shape,X_test4.shape
```

Out[28]:

```
((35000, 200), (15000, 200))
```

In [0]:

```
pickle.dump(X_train4, open('X_train4.p', 'wb'))
pickle.dump(X_test4, open('X_test4.p', 'wb'))
```

In [29]:

```
########################################################

from sklearn.neighbors import KNeighborsClassifier
from sklearn.cross_validation import cross_val_score
from sklearn.metrics import make_scorer
from sklearn.metrics import f1_score


# 10 fold Cross Validation
myList = list(range(0,15))
neighbors = list(filter(lambda x: x % 2 != 0, myList))

cv_scores = []

for k in neighbors:
    knn = KNeighborsClassifier(n_neighbors=k,algorithm='kd_tree')
    scores = cross_val_score(knn,X_train4,y_train, cv=10, scoring='f1_weighted')
    print('\nThe cross validation score for K = {} is {}.'.format(k,scores.mean()))
    cv_scores.append(scores.mean())

MSE = [1 - x for x in cv_scores]

optimal_k = neighbors[MSE.index(min(MSE))]
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/cross_validation.py:41: DeprecationWarning: This mo
dule was deprecated in version 0.18 in favor of the model_selection module into which all the refa
ctored classes and functions are moved. Also note that the interface of the new CV iterators are d
ifferent from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

```
The cross validation score for K = 1 is 0.8482374378609798.

The cross validation score for K = 3 is 0.8631362935743905.

The cross validation score for K = 5 is 0.8669735990764978.

The cross validation score for K = 7 is 0.8649125187106044.

The cross validation score for K = 9 is 0.8654870072245856.

The cross validation score for K = 11 is 0.8635436858777655.

The cross validation score for K = 13 is 0.8626279400870545.
```

In [30]:

```
print('\nThe optimal number of neighbors is %d.' % optimal_k)
```

```
The optimal number of neighbors is 5.
```

In [0]:

```
from sklearn.neighbors import KNeighborsClassifier
knn_optimal = KNeighborsClassifier(n_neighbors=5,algorithm='brute')

knn_optimal.fit(X_train4,y_train)

pred = knn_optimal.predict(X_test4)
```

In [32]:

```
f1 = f1_score(y_test, pred, average='micro')

f1_sc =f1*100

print('\nThe f1 score of the knn classifier of BOW for k = %d is %f%%' % (5, f1_sc))
```

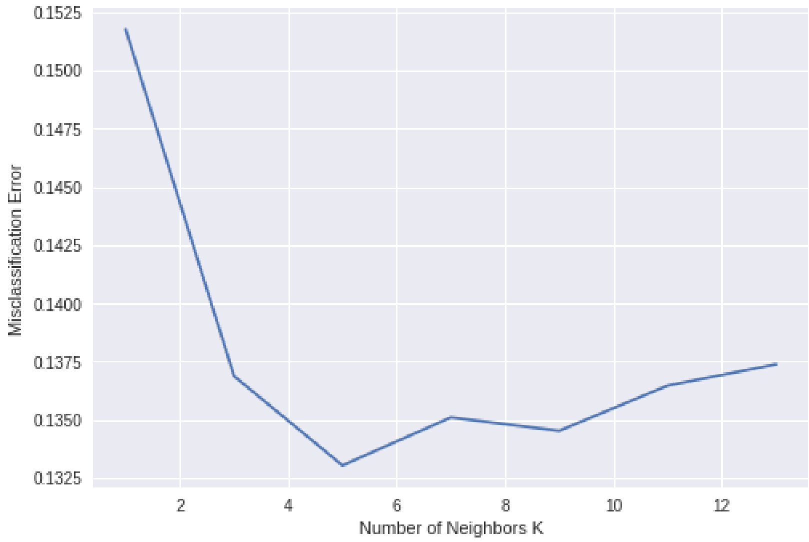
```
The f1 score of the knn classifier of BOW for k = 5 is 89.113333%
```

In [33]:

```
import matplotlib.pyplot as plt
import numpy as np
# plot misclassification error vs k
plt.plot(neighbors, MSE)


plt.xlabel('Number of Neighbors K')
plt.ylabel('Misclassification Error')
plt.show()

print("the misclassification error for each k value is : ", np.round(MSE,3))
```




```
the misclassification error for each k value is :  [0.152 0.137 0.133 0.135 0.135 0.136 0.137]
```

Observation:

==

- F1 score of BOW at hyperparameter K = 3

    83.53%

- F1 score of TFIDF at hyperparameter K = 3

    87.45%

- F1 score of AVG W2V at hyperparameter K=5

    88.82%

- F1 score of TFIDF W2V at hyperparameter k=5

    89.11%

==